Airport Robot

Welcome to Airport Robot on Exercism's Go Track. If you need help running the tests or submitting your code, check out `HELP.md`. If you get stuck on the exercise, check out `HINTS.md`, but try and solve it without using those first :)

Introduction

Interface as a set of methods

In its simplest form, an **interface type** is a set of method signatures. Here is an example of an interface definition that includes two methods `Add` and `Value`:

```
type Counter interface {
    Add(increment int)
    Value() int
}
```

The parameter names like `increment` can be omitted from the interface definition but they often increase readability.

Interface names in Go do not contain the word `Interface` or `I`. Instead, they often end with `er`, e.g. `Reader`, `Stringer`.

Implementing an interface

Any type that defines the methods of the interface automatically implicitly "implements" the interface. There is no `implements` keyword in Go.

The following type implements the `Counter` interface we saw above.

```
type Stats struct {
    value int
    // ...
```

```
}

func (s Stats) Add(v int) {
    s.value += v
}

func (s Stats) Value() int {
    return s.value
}

func (s Stats) SomeOtherMethod() {
    // The type can have additional methods not mentioned in the interface.
}
```

For implementing the interface, it does not matter whether the method has a value or pointer receiver. (Revisit the methods concepts if you are unsure about those.)

> A value of interface type can hold any value that implements those methods. [^1]

That means `Stats` can now be used in all the places that expect the `Counter` interface.

```
func SetUpAnalytics(counter Counter) {
    // ...
}

stats := Stats{}
SetUpAnalytics(stats)
// works because Stats implements Counter
```

Because interfaces are implemented implicitly, a type can easily implement multiple interfaces. It only needs to have all the necessary methods defined.

Empty interface

There is one very special interface type in Go, the **empty interface** type that contains zero methods. The empty interface is written like this: `interface{}` . In Go 1.18 or higher, `any` can be used as well. It was defined as an alias.

Since the empty interface has no methods, every type implements it implicitly. This is helpful for defining a function that can generically accept any value. In that case, the function parameter uses the empty interface type.

Instructions

The new airport in Berlin hired developers for their robots lab and you are starting your job there. They have clunky, somewhat humanoid-looking robots that they are trying to use to improve customer service.

Your first task on the job is to write a program so that the robot can greet people in their native language after they scanned their passports at the self-check-in counter.

The robot is proud of its abilities so it will always say which language it can speak first and then greet the person. For example, if someone scans a German passport the robot would say:

```
I can speak German: Hallo Dietrich!
```

1. Create the abstract greeting functionality

You will not write the code for the different languages yourself so you need to structure your code for the robot so that other developers can easily add more languages later.

As a first step, define an interface `Greeter` with two methods.

- `LanguageName` which returns the name of the language (a `string`) that the robot is supposed to greet the visitor in.
- `Greet` which accepts a visitor's name (a `string`) and returns a `string` with the greeting message in a specific language.

Next, implement a function `SayHello` that accepts the name of the visitor and anything that implements the `Greeter` interface as arguments and returns the desired greeting string. For example, imagine a German `Greeter` implementation for which `LanguageName` returns `"German"` and `Greet` returns `"Hallo {name}!"`:

```
SayHello("Dietrich", germanGreeter)
// => "I can speak German: Hallo Dietrich!"
```

2. Implement Italian

Now your job is to make the robot work for people that scan Italian passports.

For that, create a struct `Italian` and implement the two methods that are needed for the struct to fulfill the `Greeter` interface you set up in task 1. You can greet someone in Italian with `"Ciao {name}!"` .

3. Implement Portuguese

Before you call it a day, you are also supposed to finish the functionality to greet people in Portuguese.

For that, create a struct `Portuguese` and implement the two methods that are needed for the struct to fulfill the `Greeter` interface here as well. You can greet someone in Portuguese with `"Olá {name}!"` .

Source

Created by

- @junedev